



Instituto Politécnico Nacional

ESCUELA SUPERIOR DE COMPUTO

Ingeniería en Sistemas Computacionales



Compiladores

Práctica 4

Analizador léxico

PROFESOR: RAFAEL NORMAN SAUCEDO DELGADO

ALUMNO: DANIEL MURGUÍA JIMÉNEZ

No. boleta: 2016630491

GRUPO: 3CV6

Introducción

En esta práctica se implementa un analizador léxico para el lenguaje C, esto con finalidad de desarrollar un compilador de lenguaje C a ensamblador, se seleccionó este proyecto puesto a que se desea comprender de qué manera funciona un compilador tradicional del lenguaje C al ser un claro ejemplo de la forma en que funcionan los compiladores. C es un lenguaje de programación creado en 1972 por Dennis M. Ritchie en los Laboratorios Bell como evolución del anterior lenguaje B, a su vez basado en BCPL. Al igual que B, es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones. El analizador léxico será generado en Flex; Flex es una herramienta para generar programas que reconocen patrones léxicos en un texto; Flex se encarga patrones léxicos en tokens que pueden servir para estructurar una gramática. Para esta práctica se utilizó Flex para el sistema operativo Windows, en su versión 2.5.4.

Desarrollo

Para el desarrollo de esta practico se comenzó ejemplificando el lenguaje C puesto a que el compilador a trabajar será de lenguaje C a ensamblador. Se hicieron 3 ejemplos en los que se muestra de que manera funciona el lenguaje C.

Ejemplo 1.

```
1. typedef tipo_existente nuevo_nombre;
2. typedef float real;
3. typedef struct {
4. float x, y;
5. { punto;
6.
7. int main(void){
8. float x, y
9. printf("Escriba la coordenada x\n");
10. scanf("%f",&x);
11. printf("Escriba la coordenada y\n");
12. scanf("%f",&y);
13. if (x>=0 && y>=0){
14. printf ("El punto (%f,&f) está en el primer cuadrante\n",x,y);
15. }
16. else if (x>=0 && y<0){
17. printf ("El punto (%f,&f) está en el cuarto cuadrante\n",x,y);
18. }
19. else if (y>=0 && x<0){
20. printf ("El punto (%f,&f) está en el segundo
21. cuadrante\n",x,y);
22. }
23. else if (y<0 && x<0){
24. printf ("El punto (%f,&f) está en el tercer
25. cuadrante\n",x,y);
26. }
27. }
28. }
```

Ejemplo 2.

```
1. int main(void){
2. float suma=0.0;
3. float x;
4. int n;
5. printf("Introduzca cuántos números va a sumar) ;
6. scanf("%d",&n);
7. for (i=0;i<n;i++){
8. scanf("%f",&x);
9. suma = suma + x;
10. }
11. printf("la suma de los %d números leídos vale %f",n,suma);
12. }
```

Ejemplo 3

```
1. int main(void){
2. float suma=0.0;
3. int x;
4. printf("Introduzca un número positivo, 0 para finalizar\n") ;
5. scanf("%f",&x);
6. while (x>0){
7. suma = suma + x;
8. printf("Introduzca un número positivo,0 para finalizar\n");
9. scanf("%f",&x);
10. }
11. printf("la suma de los números leídos vale %f",suma);
12. }
```

Tras el análisis de los ejemplos y de las palabras reservadas del lenguaje C, se lograron identificar 30 clases léxicas, las cuales son:

1. especificadores de tipo
2. sentencia de control
3. ciclo
4. Tipo de dato
5. calificador de tipo
6. modificador de tipo
7. salto de sección
8. palabra reservada
9. operador
10. estructura
11. identificador
12. literal entera base hexadecimal
13. literal entera base octal
14. literal entera base decimal
15. literal caracter

16. literal numérica exponencial
17. literal numérica real exponencial
18. literal de cadena
19. operador relacional
20. operador lógico
21. operador lógico bit a bit
22. operador de incremento
23. operador de decremento
24. asignación (la clase asignación cuenta con varias subclases):
 - a) asignación
 - b) asignación or
 - c) asignación derecha
 - d) asignación izquierda
 - e) asignación de suma
 - f) asignación de resta
 - g) asignación de multiplicación
 - h) asignación de división
 - i) asignación de modulo
 - j) asignación and
 - k) asignación de xor
25. elipsis
26. operador de puntero
27. operador aritmético
28. separador
29. operador condicional
30. agrupación

A cada una de estas clases léxicas, les corresponde una expresión regular que la define, listadas en el mismo orden visto anteriormente, sus expresiones regulares son:

1. "auto"|"typedef"|"static"|"extern"|"register"
2. "break"|"if"|"else"|"switch"
3. "do"|"while"|"for"
4. "char"|"double"|"float"|"int"|"enum"|"void"
5. "const"|"volatile"
6. "short"|"long"|"signed"|"unsigned"
7. "continue"|"goto"
8. "default"|"case"|"return"
9. "sizeof"
10. "struct"|"union"
11. {L}{L}{D}*
12. 0[xX]{H}+{IS}?
13. 0{D}+{IS}?

14. $\{D\}^+\{IS\}?$
15. $L?'(\backslash. | [^ \backslash])+'$
16. $\{D\}^+\{E\}\{FS\}?$
17. Para la clase literal numérica real exponencial se tienen 2 expresiones regulares:
 - a. $\{D\}^*."{D}^+(\{E\})?\{FS\}?$
 - b. $\{D\}^+."{D}^*(\{E\})?\{FS\}?$
18. $L?\backslash"(\backslash. | [^ \backslash])*"$
19. $"<"| ">"| "=="| "<="| ">="| "!="$
20. $"\&\&"| "||"$
21. $"\&"| "|"|"^"| "~"| ">>"| "<<"| "!"$
22. $"++"$
23. $"--"$
24. Para la clase asignación se tienen diversas subclases con expresiones regulares propias de cada subclase:
 - a. $"="$
 - b. $"|="$
 - c. $">="$
 - d. $"<="$
 - e. $"+="$
 - f. $"-="$
 - g. $"*="$
 - h. $" /= "$
 - i. $"%= "$
 - j. $"\&= "$
 - k. $"^= "$
25. $"..."$
26. $"->"| "."$
27. $"+"| "-"| "*"| "/"| "%"$
28. $";"| ",", "("| "{"| "<%"| "("| "}%>"$
29. $"?"| ":"$
30. $"("| ")"| "["| "]"| "<:"| "("| "":>"$

Para poder crear el analizador léxico, fue necesario codificar todas estas clases léxicas junto con sus expresiones regulares en lenguaje Lex para posteriormente utilizar Flex y generarlo. El código Lex para la descripción de este analizador léxico es el siguiente:

Código Lex de lexico.l

```

1. D      [0-9]
2. L      [a-zA-Z_]
3. H      [a-zA-F0-9]
4. E      [Ee][+-]?{D}+
5. FS     (f|F|l|L)
6. IS     (u|U|l|L)*
7.
8. %{

```

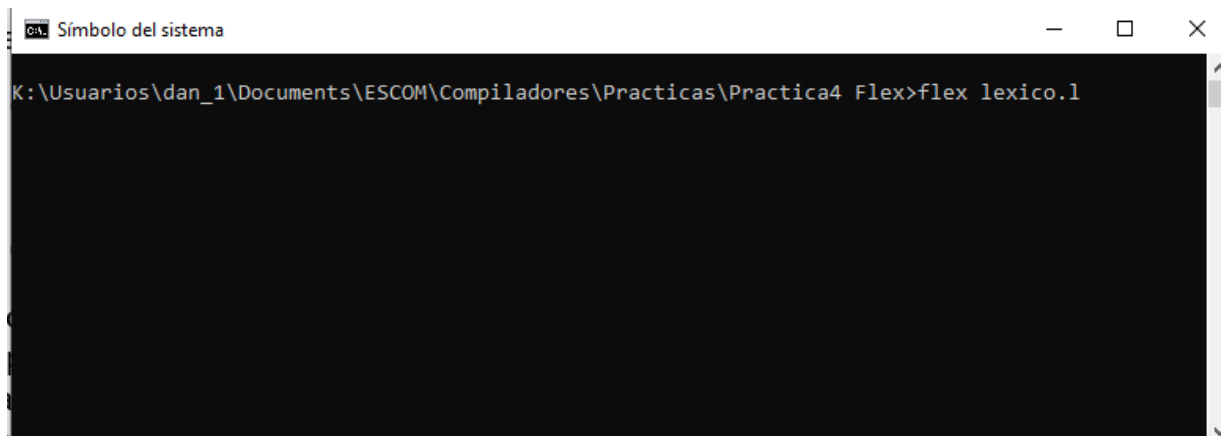
```

9. #include <stdio.h>
10.
11. %}
12.
13. %%
14. /*"                                { printf("<inicio de comentario>"); }
15.
16. "auto"|"typedef"|"static"|"extern"|"register"    { printf("<especificador de tipo>"); }
17. "break"|"if"|"else"|"switch"                    { printf("<sentencia de control>"); }
18. "do"|"while"|"for"                              { printf("<ciclo>"); }
19. "char"|"double"|"float"|"int"|"enum"|"void"      { printf("<Tipo de dato>"); }
20. "const"|"volatile"                              { printf("<calificador de tipo>"); }
21. "short"|"long"|"signed"|"unsigned"              { printf("<modificador de tipo>"); }
22. "continue"|"goto"                              { printf("<salto de seccion>"); }
23. "default"|"case"|"return"                       { printf("<palabra reservada>"); }
24. "sizeof"                                         { printf("<operador>"); }
25. "struct"|"union"                               { printf("<estructura>"); }
26. {L}({L}|{D})*                                  { printf("<identificador>"); }
27. 0[xX]{H}+{IS}?                                { printf("<literal entera base hexadecimal>"); }
28. 0{D}+{IS}?                                     { printf("<literal entera base octal>"); }
29. {D}+{IS}?                                       { printf("<literal entera base decimal>"); }
30. L?'\(\.\|[\^\\']\)+'                          { printf("<literal caracter>"); }
31. {D}+{E}{FS}?                                   { printf("<literal numerica exponencial>"); }
32. {D}*"."{D}+({E})?{FS}?                         { printf("<literal numerica real exponencial>"); }
33. {D}+("."{D}*({E})?{FS})?                       { printf("<literal numerica real exponencial>"); }
34. L?\("\.\.\|[\^\\"]\)*\                          { printf("<literal de cadena>"); }
35. "<"|">"|"=="|"<="|">="|"!="                  { printf("<operador relacional>"); }
36. "&&"|"||"                                     { printf("<operador logico>"); }
37. "&"|"|"|"^"|"~"|">>"|"<<"|"!"              { printf("<operador logico bit a bit>"); }
38. "++"                                           { printf("<operador de incremento>"); }
39. "--"                                           { printf("<operador de decremento>"); }
40. "|="                                           { printf("<asignacion or>"); }
41. "...                                           { printf("<elipsis>"); }
42. ">>="                                           { printf("<asignacion derecha>"); }
43. "<<="                                           { printf("<asignacion izquierda>"); }
44. "+="                                           { printf("<asignacion de suma>"); }
45. "-="                                           { printf("<asignacion de resta>"); }
46. "*="                                           { printf("<asignacion de multiplicacion>"); }
47. "/="                                           { printf("<asignacion de division>"); }
48. "%="                                           { printf("<asignacion de modulo>"); }
49. "&="                                           { printf("<asignacion and>"); }
50. "^="                                           { printf("<asignacion de xor>"); }
51. "->"|".".                                       { printf("<operador de puntero>"); }
52. "+"|"-"|"*"|"/"|"%"                           { printf("<operador aritmetico>"); }
53. ";"|","|("{"| "<%"|("}"| "%>")              { printf("<separador>"); }
54. "?"|":"                                         { printf("<operador condicional>"); }
55. "="                                           { printf("<asignacion>"); }
56. "("|"")|("["| "<:"|("]"| ">:")              { printf("<agrupacion>"); }
57. [ \t\v\n\f]                                    { printf(" "); }
58. .                                              { /* ignore bad characters */ }
59.
60. %%
61.
62. yywrap()
63. {
64.     return(1);
65. }

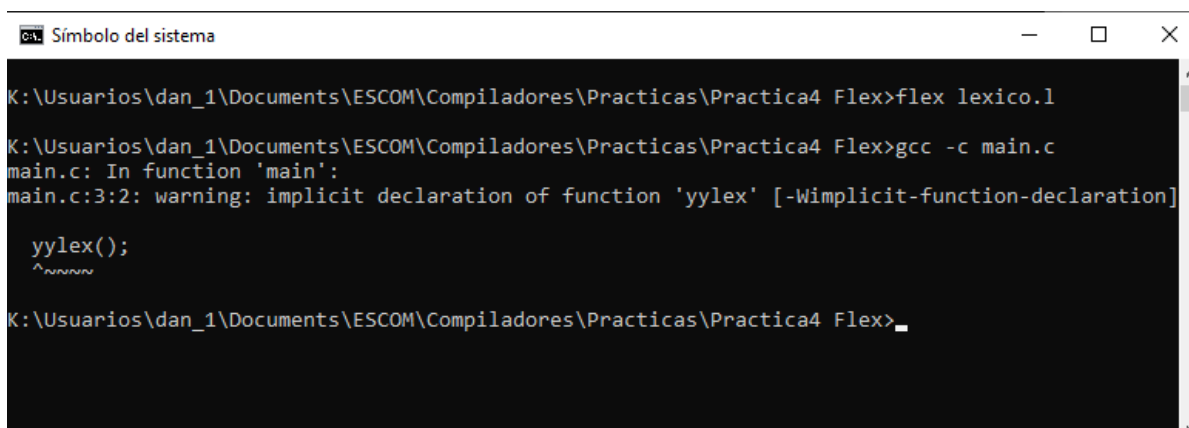
```

Finalmente tras haber encontrado las clases léxicas y haber descrito el analizador en el lenguaje Lex lo que se hizo fue compilar el archivo lexico.l con el comando "flex lexico.l" el cual nos generó un archivo lex.yy.c; este archivo generado junto con un archivo main.c, son compilados con el comando "gcc -c" lo que nos genera 2 archivos: lex.yy.o y main.o. Para que nuestro compilador de C, genere un archivo ejecutable al momento de compilar, es necesario que se enlacen los 2 archivos ".o" generados utilizando el comando "gcc main.o lex.yy.o" en el sistema operativo windows o con "gcc main.o lex.yy.o -lfl" en el sistema operativo Linux, una vez hecho esto, obtenemos un archivo ejecutable a.exe o a.out, dependiendo del sistema operativo que se esté usando. Al ejecutar este archivo, podemos introducir programas en lenguaje C los serán clasificados en las diversas clases léxicas que contengan, de acuerdo a lo descrito en el analizador léxico.

Compilación.



```
Símbolo del sistema
K:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>flex lexico.l
```



```
Símbolo del sistema
K:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>flex lexico.l
K:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>gcc -c main.c
main.c: In function 'main':
main.c:3:2: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
    yylex();
    ^~~~~~
K:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>_
```

```
Símbolo del sistema
C:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>flex lexico.l
C:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>gcc -c main.c
main.c: In function 'main':
main.c:3:2: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]

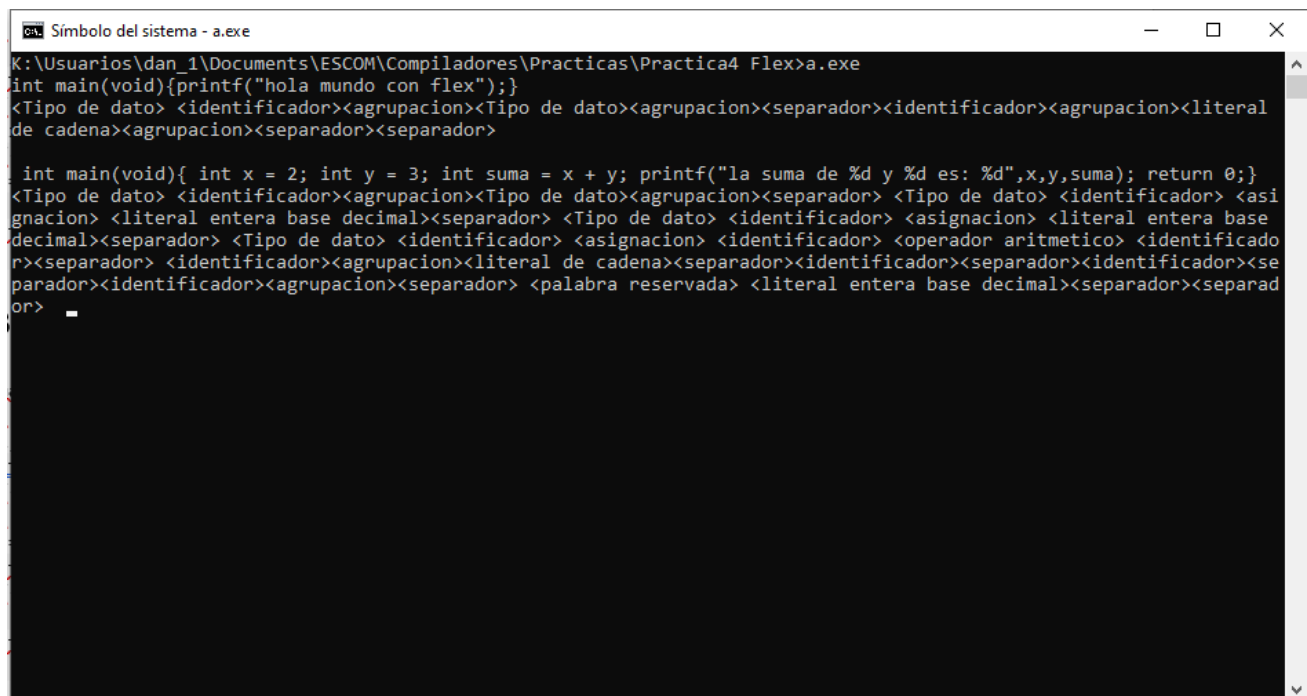
    yylex();
    ^~~~~~
C:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>gcc -c lex.yy.c
lexico.l:68:1: warning: return type defaults to 'int' [-Wimplicit-int]
yywrap()
^~~~~~
C:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>_
```

```
Símbolo del sistema
K:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>gcc -c main.c
main.c: In function 'main':
main.c:3:2: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]

    yylex();
    ^~~~~~
K:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>gcc -c lex.yy.c
lexico.l:68:1: warning: return type defaults to 'int' [-Wimplicit-int]
yywrap()
^~~~~~
K:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>gcc main.o lex.yy.o
K:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>
```

Ejecución

```
Símbolo del sistema - a.exe
K:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>a.exe
int main(void){printf("hola mundo con flex");}
<Tipo de dato> <identificador><agrupacion><Tipo de dato><agrupacion><separador><identificador>
<agrupacion><literal de cadena><agrupacion><separador><separador>
```

```
Símbolo del sistema - a.exe
K:\Usuarios\dan_1\Documents\ESCOM\Compiladores\Practicas\Practica4 Flex>a.exe
int main(void){printf("hola mundo con flex");}
<Tipo de dato> <identificador><agrupacion><Tipo de dato><agrupacion><separador> <Tipo de dato> <identificador> <asignacion> <literal entera base decimal><separador> <Tipo de dato> <identificador> <asignacion> <literal entera base decimal><separador> <Tipo de dato> <identificador> <asignacion> <identificador> <operador aritmetico> <identificador> <literal de cadena><separador><identificador><separador><identificador><separador><identificador><separador><identificador><agrupacion><separador> <palabra reservada> <literal entera base decimal><separador><separador>
...
```

Como se puede observar, al momento de introducir un programa en C a nuestro analizador léxico, este descompone las sentencias en tokens que posteriormente asigna a la clase léxica correspondiente.

Conclusión

Lo que pudimos observar durante el desarrollo de esta práctica es que una parte importante del proceso de compilación de cualquier lenguaje es el analizador léxico. Este nos permite conocer la manera en que un programa escrito en cierto lenguaje está estructurado y gracias a la utilización de la herramienta Flex, nos es posible describir las clases léxicas de un lenguaje en lenguaje Lex para posteriormente observar las clases que contenga cualquier programa escrito en el lenguaje correspondiente.

Gracias a que el lenguaje C es bastante conocido y estudiado, se convierte en un claro ejemplo de como funciona esta parte del compilador de cualquier lenguaje, permitiéndonos entender como es que el compilador de cierto lenguaje, reconoce la composición de un programa.

Referencias

Kernighan, B., & Ritchie, D. (1988). *C Programming Language* (2nd ed.). Pearson.

C. (s. f.). *Manual de flex - generador de analizadores léxicos rápidos*. Ubuntu.com.

<http://manpages.ubuntu.com/manpages/bionic/es/man1/flex.1.html>

Palabras reservadas ISO/ANSI C. (s. f.). mhe.

[https://www.mhe.es/universidad/informatica/844814645X/archivos/general_apendice4.p](https://www.mhe.es/universidad/informatica/844814645X/archivos/general_apendice4.pdf)

[df](#)